Electronic Version
Stylesheet Version v1.1.1

Claims

[c1] 1. A computer-implemented system for generating and verifying the program adhering to given specifications. The said system will henceforth be referred to as SpecProc. The system consists of apparatus for doing the task as well as the method of doing the task.

[c2] 2. The apparatus of claim 1 consisting of:
The syntactical additions made to the program.
The SpecProc apparatus.

[c3] 3. The syntactical additions of claim 2 comprising:
Code elements: consisting of usual programming language syntax modified to allow embedding of assertion statements.
Assertion elements: consisting of logical definitions and assertion statements.

[c4] 4. The said logical definitions of claim 3 further comprising:
Logical function definitions
Logical predicate definitions.

[c5] 5. The said assertion statements of claim 3 further comprising of:

Embedded code statements.
Set of logical formulas.
State transformer assertions.

[c6] 6. The SpecProc apparatus of claim 2 further comprising of following subsystems:
A subsystem called library.
A subsystem called parser.
A subsystem called library interface.
A subsystem called semantic analyzer and assertion validator (henceforth referred as SAAV).
A subsystem called logic engine.
A subsystem called emitter.

[c7] 7. The said library subsystem of claim 6 consisting of:
Store for the code elements as claimed in claim 3 in compiled form including the input-output specifications.
Store for the logical definitions as claimed in claim 4 in complied form.

[c8] 8. The library subsystem of claim 7, can further optionally store all the assertion statements in the program.

[c9] 9. The input-output specifications of claim 7, which can either be partial or total.

[c10] 10. The said library subsystem of claim 7, which can either be an extension of already existing standards or can

be totally new format.

[c11] 11. The said parser subsystem of claim 6, for parsing the program, with the syntax as given in claim 3.

[c12] 12. The parse tree representation of the program's syntactical additions as claimed in claim 3.

[c13] 13. The said library interface subsystem of claim 6, consisting of routines for reading the library of claim 7, for resolving the external references in the program.

[c14] 14. The said library interface subsystem of claim 13, further comprising of ability to search the library of claim 7, based on the input-output specifications of claim 9.

[c15] 15. The external references of claim 13, consisting of:
Code elements as given in claim 3.
Logical definitions as given in claim 4.

[c16] 16. The said subsystem SAAV of claim 6, reading the parse tree of claim 12, and doing the following steps:
Resolving the external references using library interface subsystem of claim 13.
Doing the semantic analysis to determine if the parse tree of claim 12 is semantically correct and hence compliable to an executable or to a library of claim 7.

[c17] 17. State Transformation Graph (henceforth referred as

STG) consisting of logical formulas in assertion statements of claim 5 as nodes and code statements as edges, including embedded code statements of claim 5.

[c18] 18. The STG of claim 17, further consisting of edges for state transformer assertions of claim 5.

[c19] 19. The abstract values as given in the detailed description and its use in STG of claim 17.

[c20] 20. The said SAAV of claim 16, further comprising of constructing STG for each code function from the parse tree of claim 12, including the use of embedded code statements of claim 5.

[c21] 21. The said SAAV of claim 16, further comprising of:
allocating abstract memory for code and assertion variables, and
creating and assigning abstract values of claim 19 .

[c22] 22. The parse tree of claim 12, enriched with external references of claim 15, henceforth referred to as resolved parse tree.

[c23] 23. The said SAAV as in claim 16, further comprising of:
step of enriching the parse tree of claim 12 to resolved parse tree of claim 22.

[c24] 24. The said logic engine subsystem of claim 6, reading

the STG of claim 17 and doing the following steps:
For each edge proving the target node assertions using the source node assertions and code statement on the edge.
Add edges and nodes to STG of claim 17, in case logic engine encounters state transformer assertions of claim 5.

[c25] 25. The said logic engine subsystem of claim 24, further comprising of: using the defined logical function definitions and logical predicate definitions of claim 4, in the library of claim 7, through library interface of claim 13, for the proving.

[c26] 26. The said logic engine subsystem of claim 24, further comprising of: using the abstract values of claim 19, for the proving.

[c27] 27. The said logic engine subsystem of claim 24, further comprising of: prompting the human user for assistance in proving, if need be.

[c28] 28. The STG of claim 17 after being processed in logic engine of claim 24, and having new edges in place of state transformer assertions of claim 5, henceforth referred as verified STG.

[c29] 29. The said logic engine further comprising of: opti-

mization of the verified STG of claim 28.

[c30] 30. The said emitter subsystem of claim 6, reading the resolved parse tree of claim 22 and verified STG of claim 28 and doing:
Creating either library of claim 7 or executable using resolved parse tree of claim 22 and verified STG of claim 28.
Emitting input-output specifications of claim 9, for code functions.
Emitting logical definitions of claim 4 as part of the emitted library of claim 7 or executable.

[c31] 31. The said emitter subsystem of claim 30, further consisting of: Emitting the assertion statements including embedded statements of claim 5, optionally depending on users choice.

[c32] 32. All the said subsystems as in claim 6, further comprising of facility of showing errors.

[c33] 33. The method of searching the library of claim 7, by the library interface of claim 13, as claimed in claim 14.

[c34] 34. The method of reading the library subsystem of claim 7, by the library interface subsystem of claim 13, as claimed in claim 13.

[c35] 35. The method of resolving the logical definitions of claim 4, in library of claim 7, using the library interface of claim 13, as claimed in claim 23.

[c36] 36. The method of semantic analysis by SAAV of claim 16, of assertion elements of claim 3, using the parse tree of claim 12, as claimed in claim 16.

[c37] 37. The method of creating abstract values of claim 19, by SAAV of claim 16, as claimed in claim 21.

[c38] 38. The method of construction of STG of claim 17, by SAAV of claim 16, as given in claim 20.

[c39] 39. The method of claim 38, further comprising of use of embedded statements of claim 5.

[c40] 40. The method of enriching parse tree of claim 12, to resolved parse tree of claim 22, as given in claim 23.

[c41] 41. The method of using STG of claim 17, in logic engine of claim 24, as given in claim 24.

[c42] 42. The method of using abstract values of claim 19, in logic engine of claim 24, as given in claim 26.

[c43] 43. The method of adding nodes and edges to STG of claim 17, by the logic engine of claim 24, as given in claim 24.

[c44] 44. The method of optimizing the verified STG of claim 28, by the logic engine of claim 24, as given in claim 29.

[c45] 45. The method of emitting input-output specifications of claim 9 of code functions, into executable or library of claim 7, by the emitter of claim 30, as claimed in 30.

[c46] 46. The method of emitting logical definitions of claim 4, into executable or library of claim 7, by the emitter of claim 30, as claimed in 30.

[c47] 47. The method of optionally emitting the assertion statements including embedded statements of claim 5, into the executable or library of claim 7, by emitter of claim 30, as claimed in 31.